Knockout.js

Use Knockout.js to design and build dynamic client-side web applications that are extremely responsive and easy to maintain. This example-driven book shows you how to use this lightweight JavaScript framework and its Model-View-ViewModel (MVVM) pattern. You’ll learn how to build your own data bindings, extend the framework with reusable functions, and work with a server to enhance your client-side application with persistence. In the final chapter, you’ll build a shopping cart to see how everything fits together.

If you’re a web developer with experience in JavaScript, HTML, and CSS, you’re ready for Knockout.

- Learn how to create a ViewModel
- Bind HTML data and attributes, and CSS classes and styles
- Understand data binding in Knockout’s context hierarchy
- Use properties that change dynamically through user interaction
- Work with forms by using several different bindings
- Bind multiple ViewModels on a single page
- Extend or attach custom functions to observables
- Perform server-side interactions with jQuery
- Map a JavaScript object or apply JSON data to a new object

Jamie Munro has been developing websites and web applications for over 15 years. For the past six years, he’s been mentoring younger developers to enhance their web development skills. Jamie’s website (http://www.webistrate.com) provides examples to help web developers expand their experience.

“This book is a great resource for any developer looking to make their web application user experience pop. It presents the features and benefits of Knockout.js with extensive examples and in-depth explanations. You also benefit from the author’s recommendations and advice coming from lessons learned through his experience building web applications.”

— Steven Kennedy
Senior Software Developer at Fusebill
Knockout.js

Use Knockout.js to design and build dynamic client-side web applications that are extremely responsive and easy to maintain. This example-driven book shows you how to use this lightweight JavaScript framework and its Model-View-ViewModel (MVVM) pattern. You'll learn how to build your own data bindings, extend the framework with reusable functions, and work with a server to enhance your client-side application with persistence. In the final chapter, you'll build a shopping cart to see how everything fits together.

If you're a web developer with experience in JavaScript, HTML, and CSS, you're ready for Knockout.

- Learn how to create a ViewModel
- Bind HTML data and attributes, and CSS classes and styles
- Understand data binding in Knockout's context hierarchy
- Use properties that change dynamically through user interaction
- Work with forms by using several different bindings
- Bind multiple ViewModels on a single page
- Extend or attach custom functions to observables
- Perform server-side interactions with jQuery
- Map a JavaScript object or apply JSON data to a new object

Jamie Munro has been developing websites and web applications for over 15 years. For the past six years, he's been mentoring younger developers to enhance their web development skills. Jamie's website (http://www.webistrate.com) provides examples to help web developers expand their experience.

“This book is a great resource for any developer looking to make their web application user experience pop. It presents the features and benefits of Knockout.js with extensive examples and in-depth explanations. You also benefit from the author’s recommendations and advice coming from lessons learned through his experience building web applications.”

—Steven Kennedy
Senior Software Developer at Fusebill
Want to read more?

You can [buy this book](http://oreilly.com) at [oreilly.com](http://oreilly.com) in print and ebook format.

**Buy 2 books, get the 3rd FREE!**

Use discount code: OPC10

All orders over $29.95 qualify for [free shipping](http://oreilly.com) within the US.

It’s also available at your favorite book retailer, including the iBookstore, the [Android Marketplace](http://oreilly.com), and [Amazon.com](http://oreilly.com).
Knockout.js
Building Dynamic Client-Side Web Applications

Jamie Munro
# Table of Contents

## Preface

1. **Getting Started with KnockoutJS**
   - Data Binding Syntax
   - Data Binding Example
   - What Is MVVM
   - Creating a ViewModel
   - Object-Oriented ViewModels
   - ViewModels with Parameters

2. **The Ins and Outs of Data Binding**
   - Binding HTML Data
   - Binding HTML Attributes, CSS Classes, and CSS Styles
   - Condition-Based Data Binding

3. **Understanding the Data Binding Context**
   - foreach Binding
   - foreach Callbacks (Events)
   - with Binding

4. **Dynamically Changing Properties**
   - Defining an Observable
   - pureComputed Observables
   - Showing and Hiding Elements
   - Adding and Removing Elements
   - When to Use Observables
KnockoutJS is an open source JavaScript library. It was built to allow you to create dynamic and rich web applications. It is built with the Model-View-ViewModel (MVVM) pattern. Knockout makes it really simple to implement a complex user interface that responds to user interactions.

I like Knockout because it is one of the most lightweight JavaScript libraries available today. It also doesn't try to be an all-in-one framework. It serves a single purpose: data binding your ViewModel to your user interface.

Implementing Knockout involves three distinct things: a view that contains HTML and CSS elements that get data-bound to it, a ViewModel that contains the data to bind to the view, and telling Knockout to perform the data binding to the view with the ViewModel.

The examples in this chapter demonstrate how to create an HTML page with a basic ViewModel. After we review the basic data binding syntax, we will explore the various types of ViewModels that work with the view.

**Data Binding Syntax**

Data binding is accomplished by adding an HTML attribute called `data-bind` to any HTML element that you want Knockout to replace with information from your ViewModel.

Sometimes an HTML tag does not work, so Knockout also allows you to specify data bindings with HTML comments, as shown in Example 1-1.
Example 1-1. Knockout bindings using HTML comments

<!--[ko]><!-- /ko-->

Binding through HTML comments is extremely convenient when you want to wrap your binding around a block of HTML code, or you do not want to create an extra HTML object for the sole purpose of housing a data binding.

Data Binding Example

One of the most common things that Knockout is used for is to dynamically display text or HTML based on data contained within a ViewModel.

Example 1-2 shows creating a header that will display the name property within the ViewModel.

Example 1-2. Data binding text

```html
<!DOCTYPE html>
<html>
<head>
  <title>Data Binding with KnockoutJS</title>
</head>
<body>
  <h1>Hello <span data-bind="text: name"></span></h1>

  <script type='text/javascript' src='js/knockout-3.2.0.js'></script>
  <script>
    var viewModel = function() {
      this.name = 'Steve Kennedy';
    };

    ko.applyBindings(viewModel);
  </script>
</body>
</html>
```

When this example is executed in a browser, it outputs Hello Steve Kennedy inside an h1 tag.

A span tag is data-bound to the name property of the ViewModel. This is done by placing text: name inside the data-bind HTML attribute.

As mentioned in the introduction to this chapter, implementing Knockout requires three distinct things. The first is the view, which in this example is the HTML that contains the h1 and span tags that identify the data binding.
The second is the ViewModel, which in this example is the JavaScript variable/function called `viewModel` that contains a single variable name.

The third is telling Knockout to perform the data binding of the view and the ViewModel. This is accomplished by calling the `ko.applyBindings` function with a ViewModel.

When this function is executed, Knockout processes both the view and the ViewModel. All data bindings in the view are executed and dynamically replaced with the data from the ViewModel.

Knockout does not limit you to a single ViewModel per view. Large projects or classes that are well designed for reusability are common reasons to bind multiple ViewModels to a single view. This is discussed in more detail in Chapter 6.

**What Is MVVM**

The Model-View-ViewModel (MVVM) design pattern is based largely on the Model-View-Controller (MVC) pattern. In fact, the MV is shared between them. It’s the ViewModel that really separates the two.

MVVM was designed to implement data binding between your ViewModel and your View. This is what KnockoutJS does for us and does very well. It is accomplished using some simple-to-implement HTML attributes and a JavaScript ViewModel as shown in Example 1-2.

The most important thing to remember when you are building ViewModels is that they should be organized to make it easy to represent how your View uses the data. We explore several common scenarios in the upcoming examples.

**Creating a ViewModel**

A ViewModel can be any type of JavaScript variable. In Example 1-3, let’s start with a simple JavaScript structure that contains a single property called `name`.

**Example 1-3. Basic ViewModel**

```javascript
var myFirstViewModel = {
    name: 'Steve Kennedy'
};
```

The previous example is a perfect scenario of a ViewModel that might not correlate 100% to a data model. Frequently, a data model separates the first and last name into two separate fields to make editing the data easier. However, in a view, it makes more sense to concatenate them into a single field for display.
The previous ViewModel is quite basic. ViewModels are not limited to such a simple structure, as the next examples demonstrate.

**Object-Oriented ViewModels**

Typically, when I create ViewModels, I create simple or complex JavaScript classes that allow me to leverage an object-oriented style of programming (functions, properties, abstraction, etc.).

**Object-Oriented Programming with JavaScript**

JavaScript is a fully object-oriented programming (OOP) language based on prototyping. It doesn't contain class statements like C++, C#, or PHP; however, JavaScript functions can be used to simulate the same behavior.

It also offers full support of OOP language features such as namespaces, objects, properties, inheritance, abstraction, etc.

If you are new to JavaScript or object-oriented programming, the Mozilla Developer Network (MDN) offers a great introductory article.

In Example 1-4, I create a ViewModel that is built with some basic object-orientation that will provide more functionality for our view.

**Example 1-4. Object-Oriented ViewModel**

```javascript
var mySecondViewModel = function() {
    var self = this;

    self.name = 'Steve Kennedy';

    self.getName = function() {
        return self.name;
    }
};
```

In this example, my ViewModel is a JavaScript function that I've made to act like a class in OOP. It contains the same `name` variable that I created in the previous example, but this time it is encapsulated in my class and acts like a property of it.

I also added a new function called `getName` that allows me to access my `name` property without calling my `class` property directly from elsewhere in my code.
Self = This?

You may be wondering why the first line of my class is `var self = this;`. By creating a variable called `self` and assigning it the variable `this`, it provides me a property in my class that I can use inside methods of my class and easily reference other methods or properties in my class.

Even though the resulting code between Example 1-4 and Example 1-3 look quite different, in fact, they are quite similar. The name property can be accessed in the exact same way from either example, as shown in Example 1-5.

**Example 1-5. Accessing the name property**

```javascript
alert(myFirstViewModel.name);
alert(mySecondViewModel.name); // access the property name
alert(mySecondViewModel.getName()); // access the function getName
```

All three of the previous statements will alert `name`.

**ViewModels with Parameters**

In Examples 1-3 and 1-4, the name property was hardcoded to a value. In many scenarios, data like this would be populated from a different source: a database, the results of an AJAX call, etc.

In Example 1-6, the name will be populated via an input to the ViewModel.

**Example 1-6. A ViewModel with a parameter**

```javascript
function ViewModel(name) {
    var self = this;

    self.name = name;

    self.getName = function() {
        return self.name;
    };
}

var myThirdViewModel = new ViewModel('Steve Kennedy');
```

In Examples 1-3 and 1-4, the ViewModel class was assigned directly to the variable. This example is slightly different in that a function (acting like a class) called `ViewModel` is created that accepts the name property. This class is then instantiated by passing the name into the constructor.
In all future examples, I will continue to use JavaScript classes as they provide a lot more flexibility. However, if a simple JavaScript variable or structure will suffice for your view, do not feel the need to wrap your ViewModel in a class.
O’Reilly Ebooks—Your bookshelf on your devices!

When you buy an ebook through oreilly.com you get lifetime access to the book, and whenever possible we provide it to you in five, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, Android .apk, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we’ve updated the files with corrections and additions.

Learn more at ebooks.oreilly.com

You can also purchase O’Reilly ebooks through the iBookstore, the Android Marketplace, and Amazon.com.